**Data Block & import Block in terraform configuration file**

**Data Block:**

In a Terraform configuration file, a data block is used to fetch information about existing infrastructure resources. It doesn't create or modify anything; it simply reads data. Think of it as querying or looking up details about resources that were created outside of your current Terraform configuration (e.g., manually created, created by another team, or managed by a different Terraform state).

**Key Characteristics of the data Block**

1. **Read-Only:**
   * The data block is used to read information, not to create or modify resources.
2. **External Data:**
   * It retrieves or fetch data from external sources, such as cloud provider APIs, existing infrastructure, or other Terraform configurations.
3. **Dynamic:**
   * The data is fetched dynamically during the Terraform plan or apply phase.
4. **Provider-Specific:**
   * Each provider (e.g., AWS, Azure) offers its own data sources, which can be queried using the data block.

**Syntax of a data Block:**

data "<PROVIDER>\_<TYPE>" "<NAME>" {

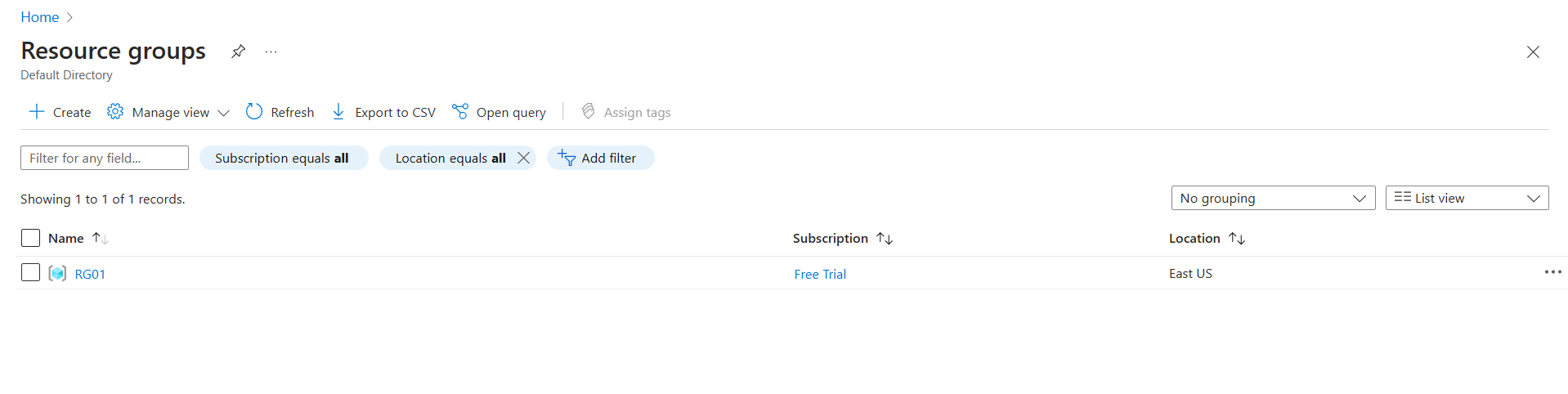
# Arguments to filter or query the data

}

* **<PROVIDER>:** The name of the provider (e.g., aws, azurerm, google).
* **<TYPE>:** The type of data source provided by the provider (e.g., ami, vpc, subnet).
* **<NAME>:** A local name for the data source, which you can use to reference the data elsewhere in your configuration.

Now let’s do it practically.

Step1: Let’s create the resource group manually from the azure portal.



**Step2:** Then fetch the details of resource group (RG01) using data block, using terraform configuration file.

#provider Block

provider "azurerm" {

  features {}

  client\_id       = "1f79e427-2ac4-4eb6-9ca0-f4dd4b3f31ee"

  client\_secret   = "Vb18Q~Zj4DMUeKgAiQ-Lpm~pj92y.7s7SLBaAcyG"

  tenant\_id       = "4a623a04-9917-4ee2-8f59-02586964c992"

  subscription\_id = "51c6d184-6756-4a9a-ade4-cd0f3d57cded"

}

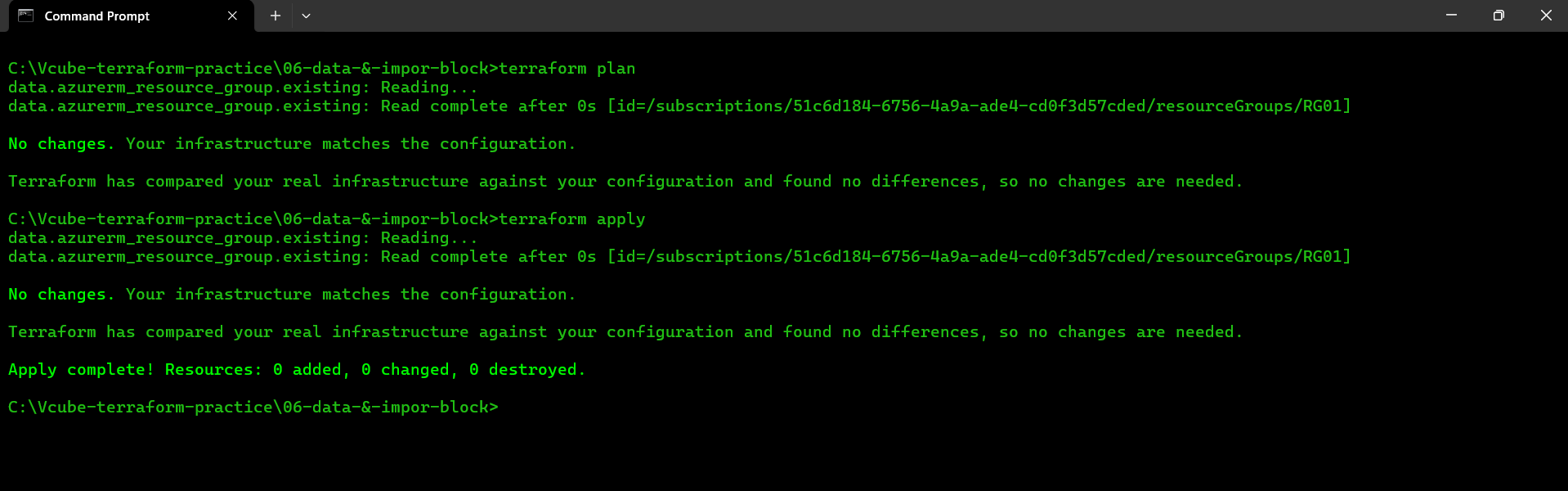
**Fig:** Main.tf file.

data "azurerm\_resource\_group" "existing" {

    name = "RG01"

}

**Fig:** Data Block.



While we do **terraform apply** it read the existing resource and provided the resource ID as show in above figure.

**Note:** We can give the output block also if you need to know clear details of existing resource in data block.

#data block

data "azurerm\_resource\_group" "existing" {

    name = "RG01"

}

#output block

output "rg\_id" {

    value = data.azurerm\_resource\_group.existing.id

}

**Step3:** Now create the Virtual Network in existing resource group (RG01) using terraform configuration file.

# virtual network block

resource "azurerm\_virtual\_network" "TFVnet" {

    name = "Vnet01"

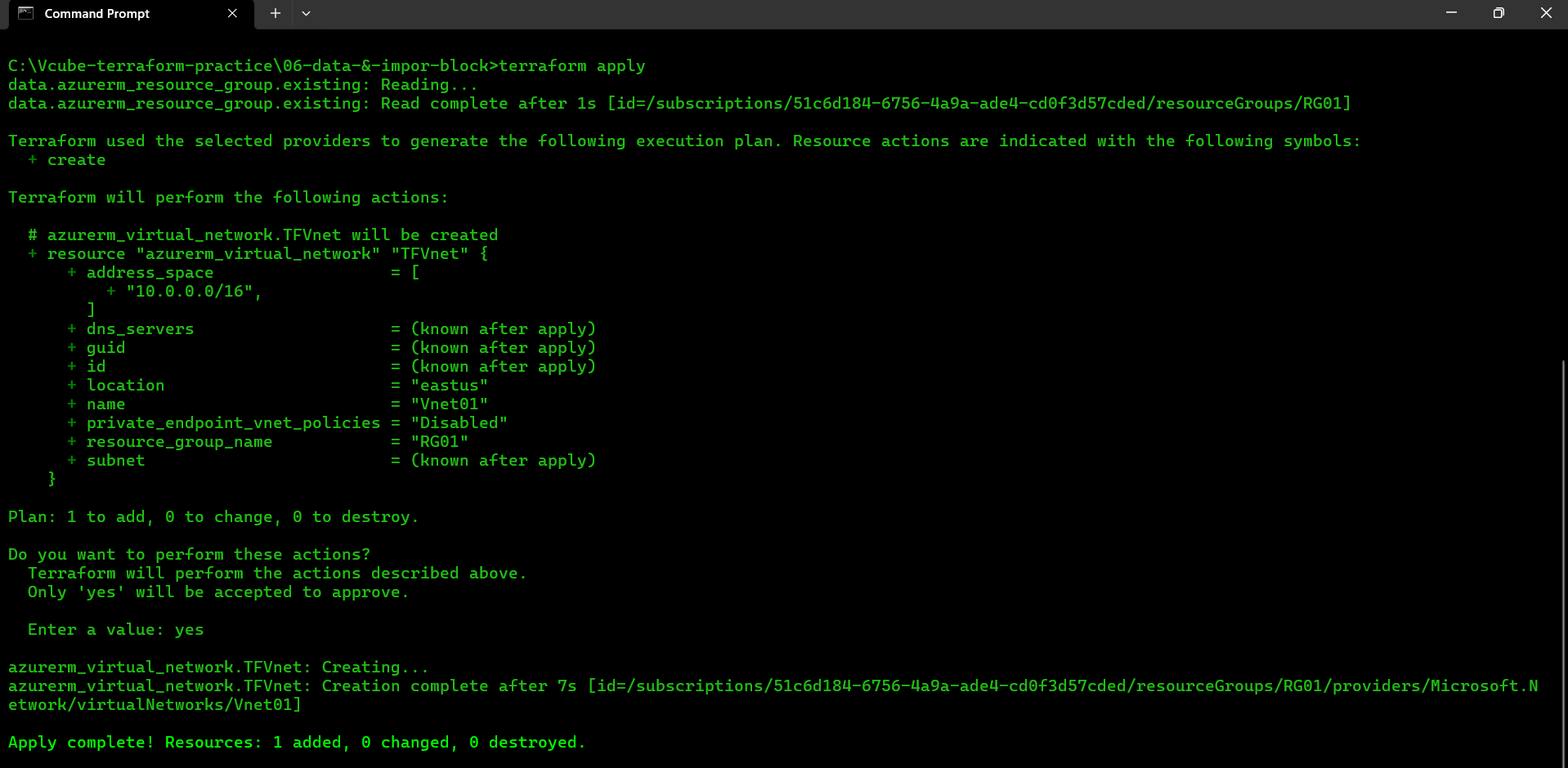
    location = data.azurerm\_resource\_group.existing.location

    resource\_group\_name = data.azurerm\_resource\_group.existing.name

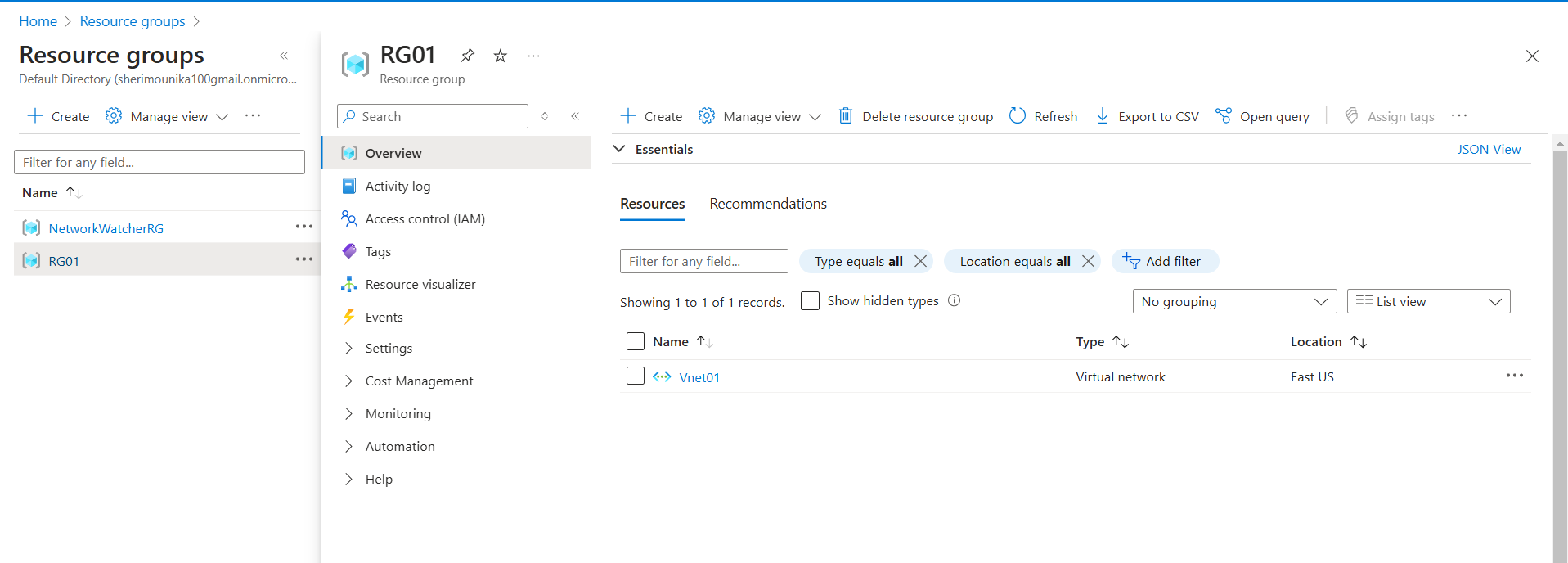
    address\_space = ["10.0.0.0/16"]

}

**Fig:** Vnet block.



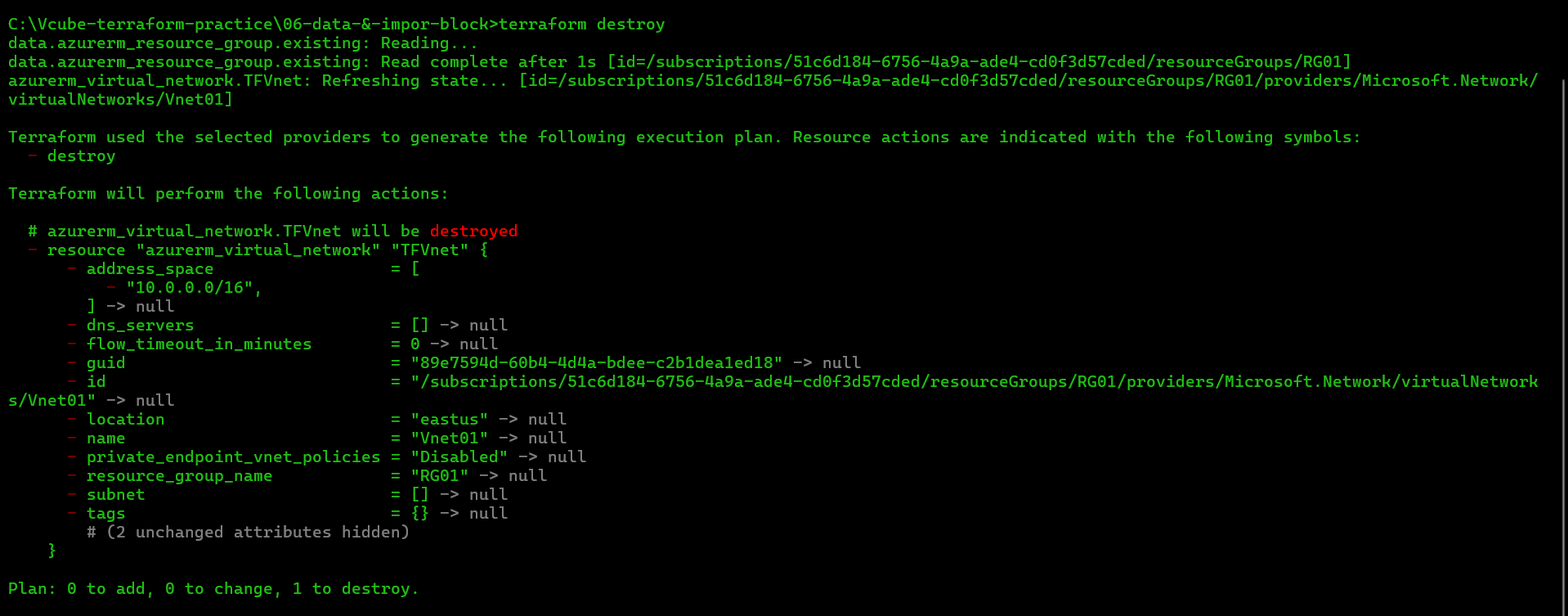
After terraform apply, only one resource group is added that is virtual network (Vnet01), and the terraform state file recognized that resource group (RG01) is already exist no need to create again.

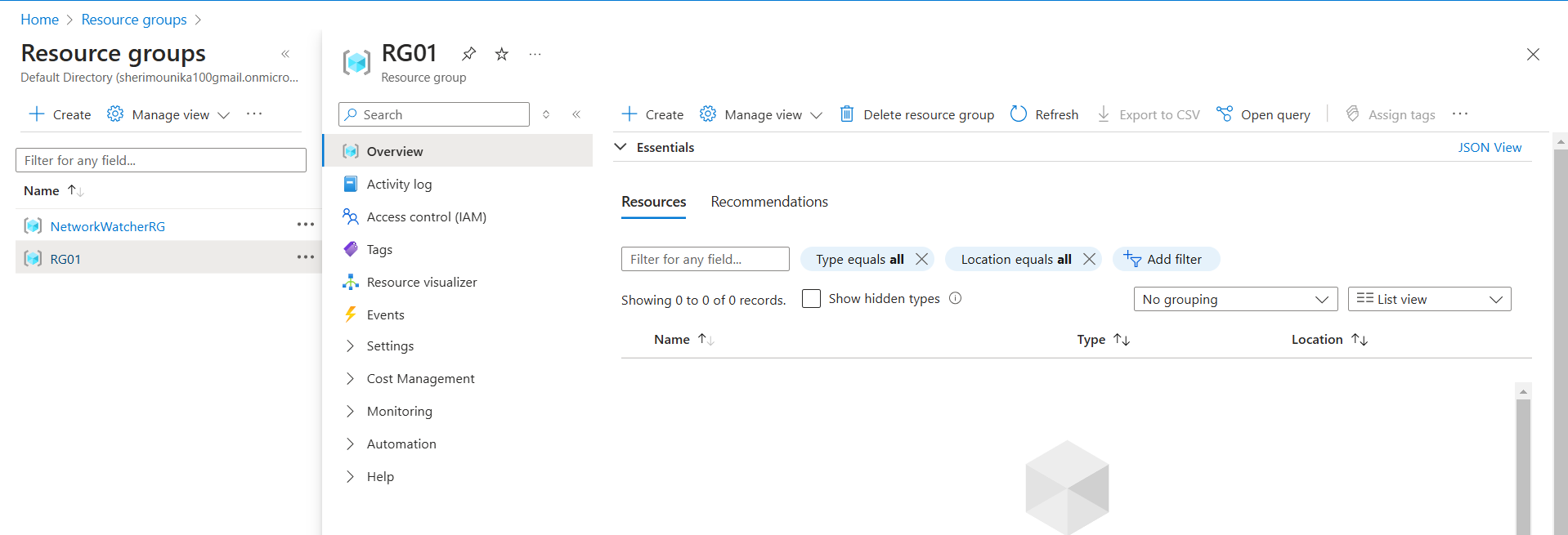


**Fig:** Vnet01 is created within the existing resource group RG01.

Here we didn’t created the resource group using terraform configuration file but we fetched it using the data block and within it we created the Virtual network.

**Note1:** The terraform.tfstate file does not consist the metadata related to the resource group (RG01). It only consist information about the virtual network (Vnet01). So as a result when we do **terraform destroy** it only delete the virtual network (Vnet01) but not the resource group (RG01).



****

**Fig:** Virtual network (Vnet01) is deleted.

**Note2:** Whatever resources we created using the terraform configuration file, these resource can only under the control of our terraform.

In order to delete the manually created resource group (RG01). First we have **to import** this resource group into our terraform.tfstate file. Then only this resource group under the control of terraform.

**Import in Terraform**

In Terraform, the **import command** is used to bring existing infrastructure resources under Terraform management. This allows you to adopt and manage resources that were created outside of Terraform (e.g., manually through a cloud provider's console or CLI) using Terraform's configuration and state management.

**Key Points About terraform import**

1. **Adopting Existing Resources:**
   * The import command is used when you already have resources created outside of Terraform and want to manage them using Terraform.
2. **State Management:**
   * The import command updates Terraforms state file to include the existing resource. However, it does not automatically generate the corresponding Terraform configuration code.
3. **Manual Configuration Required:**
   * After importing a resource, you must manually write the Terraform configuration (in **.tf** files) to match the imported resource. Otherwise, Terraform might try to recreate or destroy the resource in future operations.
4. **Syntax:**

terraform import <RESOURCE\_ADDRESS> <RESOURCE\_ID>

* + **<RESOURCE\_ADDRESS>:** The address of the resource in Terraform configuration (e.g., aws\_instance.my\_vm).
  + **<RESOURCE\_ID>:** The unique identifier of the resource in the cloud provider (e.g., an EC2 instance ID like i-0abcd1234efgh5678).

In order to import the manually created resource we have to follow two steps

step1: Create the resource block for imported resource.

step2: Import resource-ID to this resource block.

#resource block

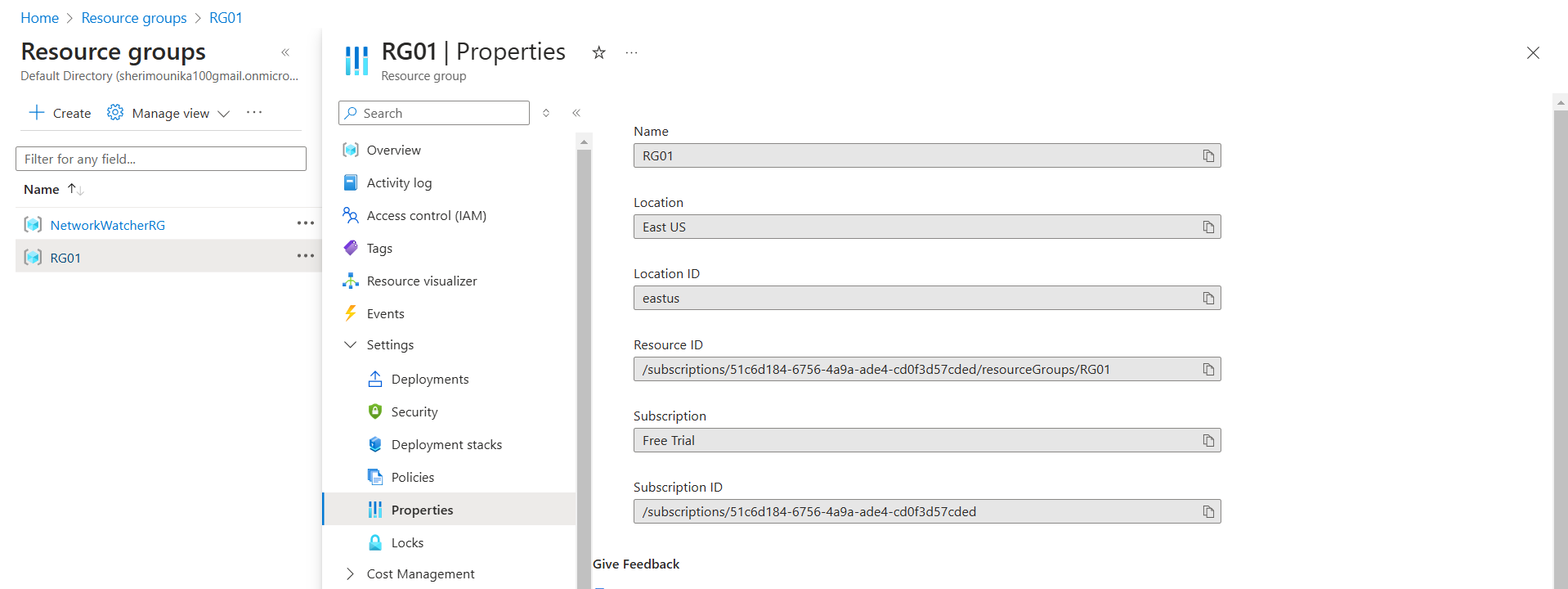
resource "azurerm\_resource\_group" "TFRG" {

    name = "RG01"

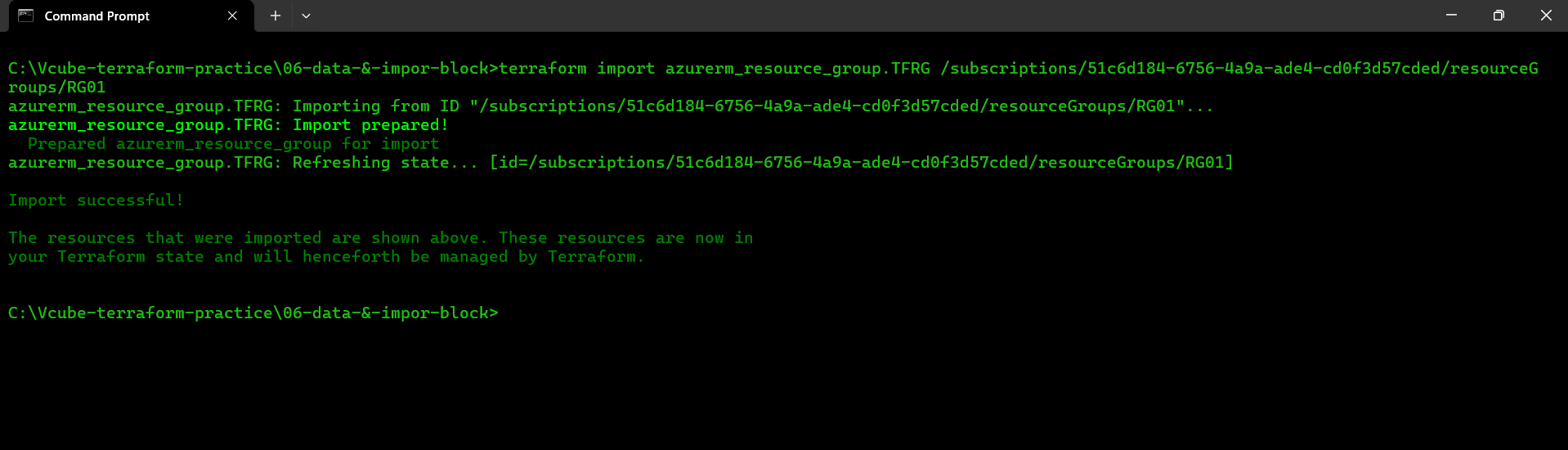
    location = "eastus"

}

Resource Block =🡺

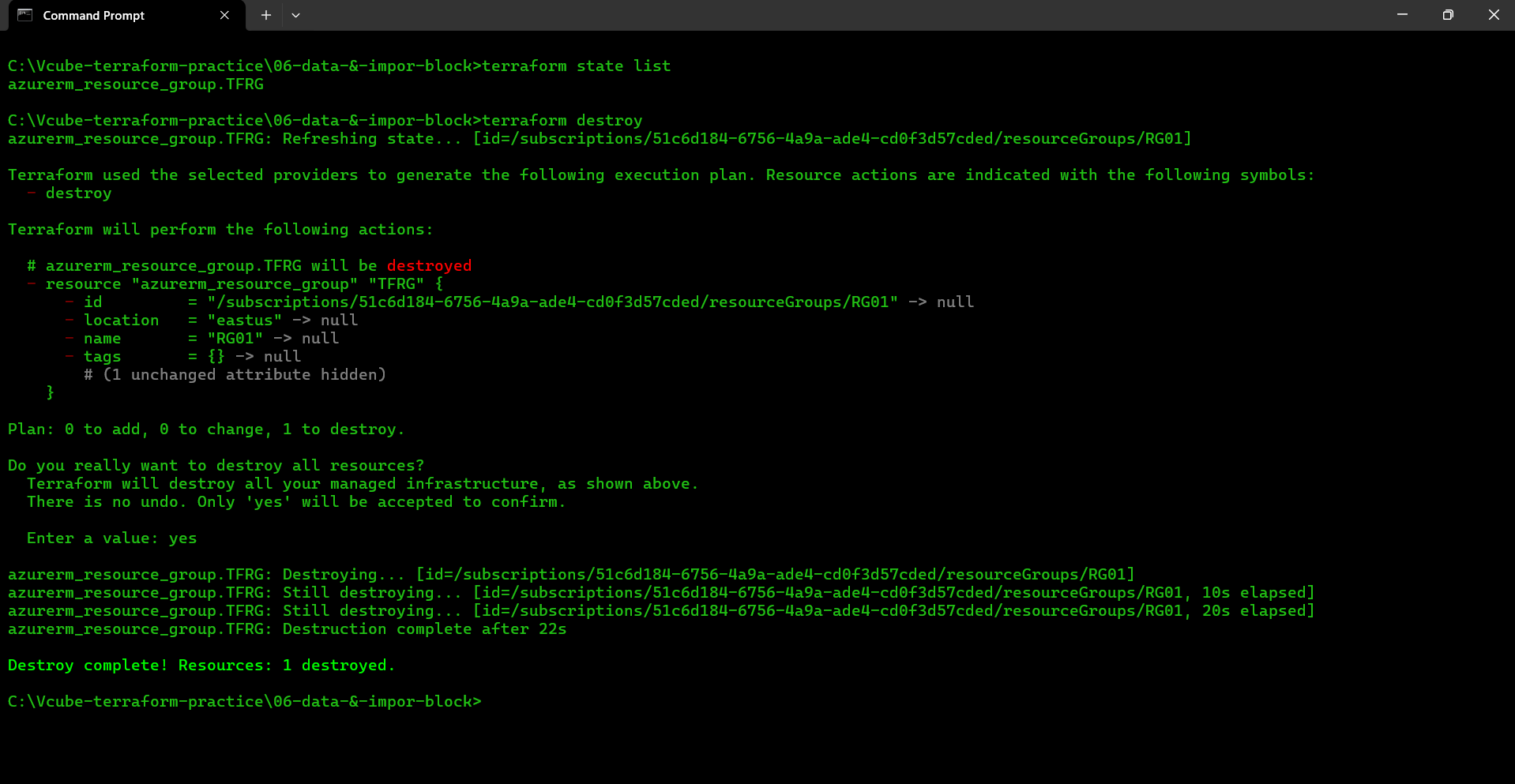


**Fig:** resource group (RG01) ID.



**Fig:** importing resource ID.

Now this resource group (RG01) is **under terraform control**. Now we can delete this resource group using **terraform destroy** command.



**Fig:** Manually created resource group (RG01) is deleted using terraform command

**Output Block in Terraform**

An **output block** in Terraform is used to display values after a successful terraform apply. It helps retrieve information about resources or computed values, making them available for reference.

**Key Characteristics of the output Block**

1. **Expose Values:**
   * Outputs make display specific values (e.g., resource IDs, IP addresses, or computed values) accessible after running terraform apply.
2. **Read-Only:**
   * Outputs are read-only and cannot be modified by other parts of the configuration.
3. **Inter-Module Communication:**
   * Outputs are commonly used to share values between Terraform modules.
4. **Human-Readable:**
   * Outputs are displayed in the Terraform CLI after applying the configuration, making them useful for debugging or providing information to users.
5. **Persistent:**
   * Output values are stored in the Terraform state file and can be queried later using the terraform output command.

**Syntax of an output Block:**

output "<NAME>" {

value = <VALUE>

description = "Description of the output"

}

* **<NAME>:** The name of the output (used to reference it later).
* **<VALUE>:** The value to expose (e.g., a resource attribute, variable, or computed value).
* **description:** (Optional) A description of the output's purpose.
* **sensitive:** (Optional) If set to true, the output value will be redacted in the CLI output.

**Let’s work with the output Block in the terraform configuration file:**

#provider Block

provider "azurerm" {

  features {}

  client\_id       = "1f79e427-2ac4-4eb6-9ca0-f4dd4b3f31ee"

  client\_secret   = "Vb18Q~Zj4DMUeKgAiQ-Lpm~pj92y.7s7SLBaAcyG"

  tenant\_id       = "4a623a04-9917-4ee2-8f59-02586964c992"

  subscription\_id = "51c6d184-6756-4a9a-ade4-cd0f3d57cded"

}

#resource group block

resource "azurerm\_resource\_group" "rg" {

  name = "RG01"

  location = "eastus"

}

**Fig:** main.tf file.

output "rg-name" {

    value = azurerm\_resource\_group.rg.name

    description = "resource group name"

}

output "rg-location" {

    value = azurerm\_resource\_group.rg.location

    description = "resource group location"

}

output "rg-id" {

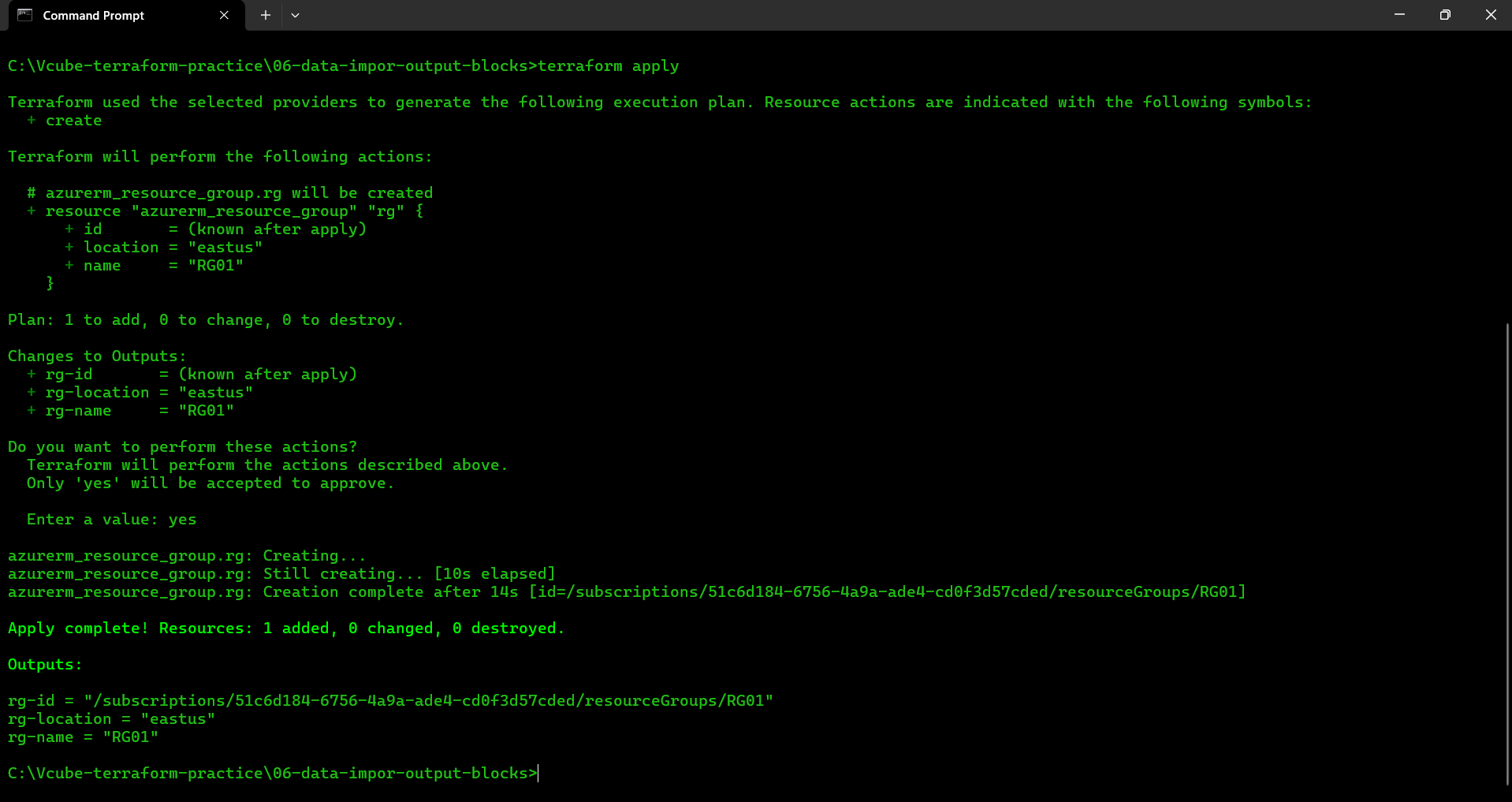
    value = azurerm\_resource\_group.rg.id

    description = "resource group id"

}

**Fig:** output block.

After doing terraform apply we can clearly see the output values which is displayed specifically as show in below figure.



**Note:** Output block is just like a print method in java or python. It is used to display the values, which is human readable format.

**“depends\_on” Argument in terraform**

In Azure Terraform (and Terraform in general), **depends\_on** is a meta-argument within a resource block that explicitly defines dependencies between resources.

It ensures that one resource or module is created or updated only after another resource or module has been successfully created or updated.

We know that in azure follows the hierarchy, that we cannot create the VM or Vnet or storage account without creating the resource group. That means VM or Vnet or Storage account depends on the resource group.

If we write the terraform code to create the resource group, virtual network and subnet at a time in a same terraform configuration file, then if we do terraform apply we will get error (no garret of execution hierarchal), because terraform don’t know the hierarchy which is to be created first and which is to be created last.

So as a result we use the “**depends\_on”** argument in resource block in order to define the dependences between the resources.

Now let’s do the above assumption or description in practically using depends\_on argument.

**Step1:** write the terraform configuration file to create the resource group, virtual network and subnet.

#resource group block

resource "azurerm\_resource\_group" "rg" {

  name = "RG01"

  location = "eastus"

}

#vertual network

resource "azurerm\_virtual\_network" "TFVnet" {

    name = "Vnet01"

    resource\_group\_name = azurerm\_resource\_group.rg.name

    location = azurerm\_resource\_group.rg.location

    address\_space = ["10.0.0.0/16"]

    depends\_on = [ azurerm\_resource\_group.rg ]

}

#subnet creation

resource "azurerm\_subnet" "TFsubnet" {

    name = "subnet01"

    virtual\_network\_name = azurerm\_virtual\_network.TFVnet.name

    resource\_group\_name = azurerm\_resource\_group.rg.name

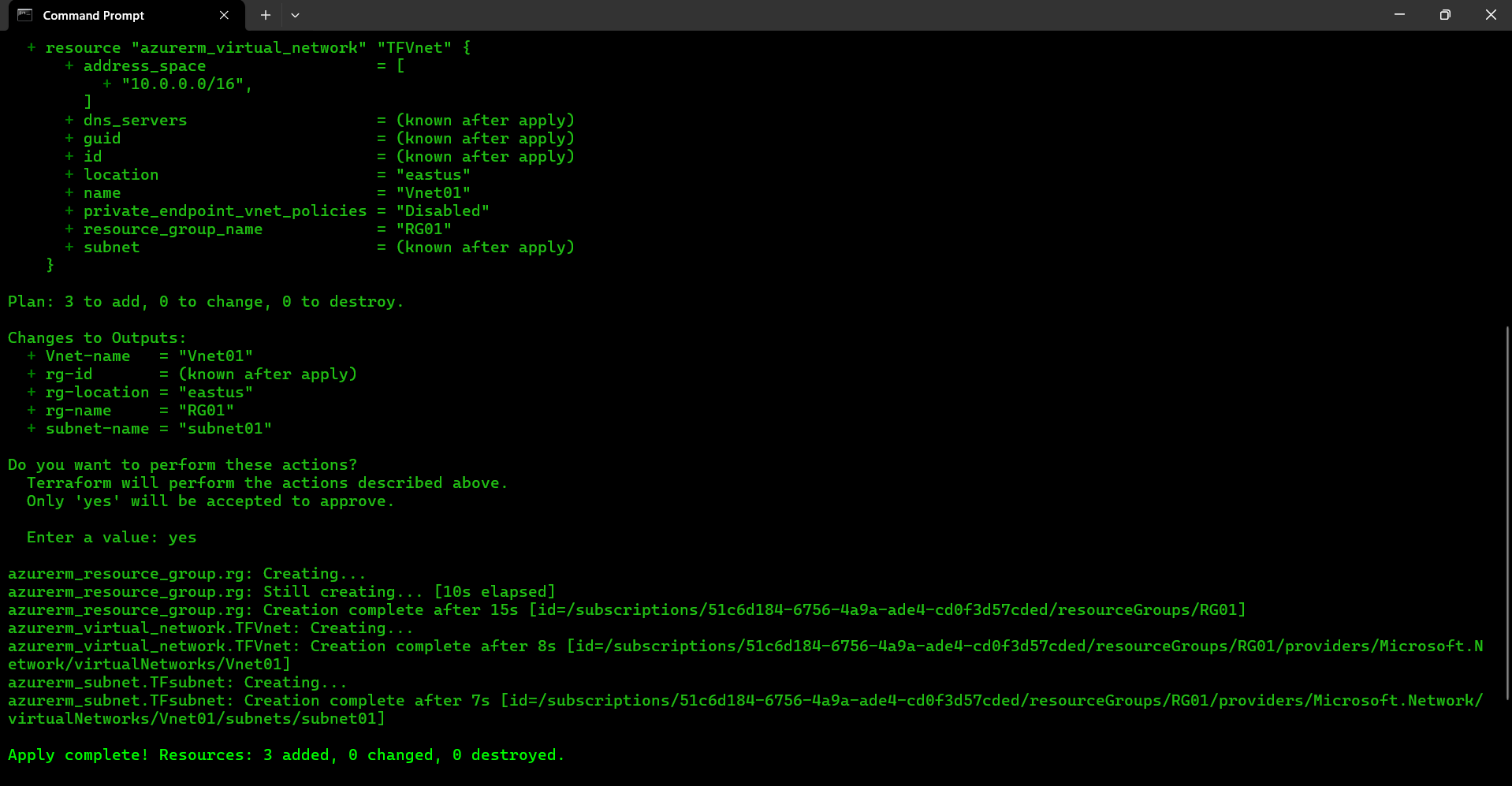
    address\_prefixes = ["10.0.1.0/24"]

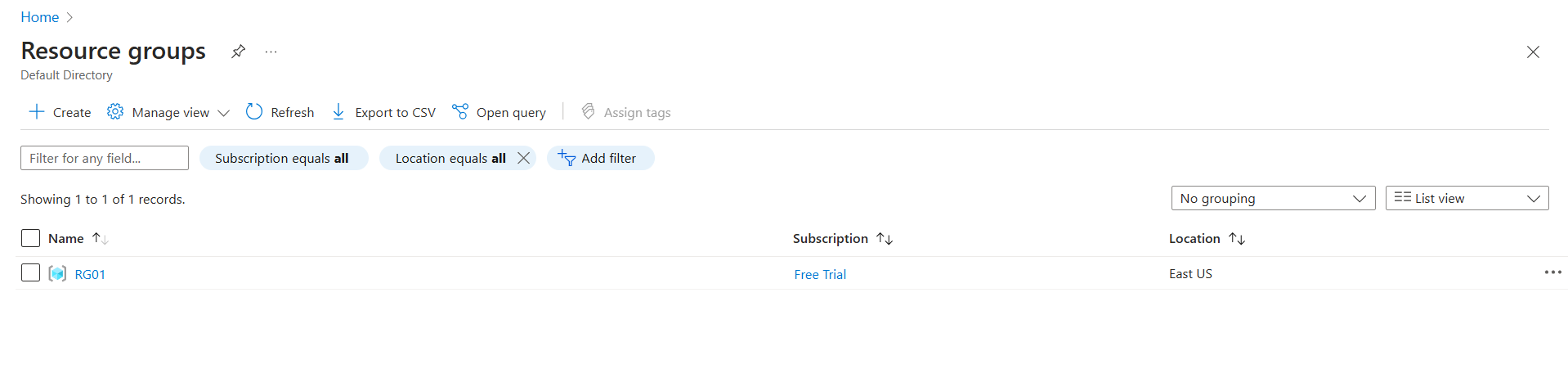
    depends\_on = [ azurerm\_virtual\_network.TFVnet ]

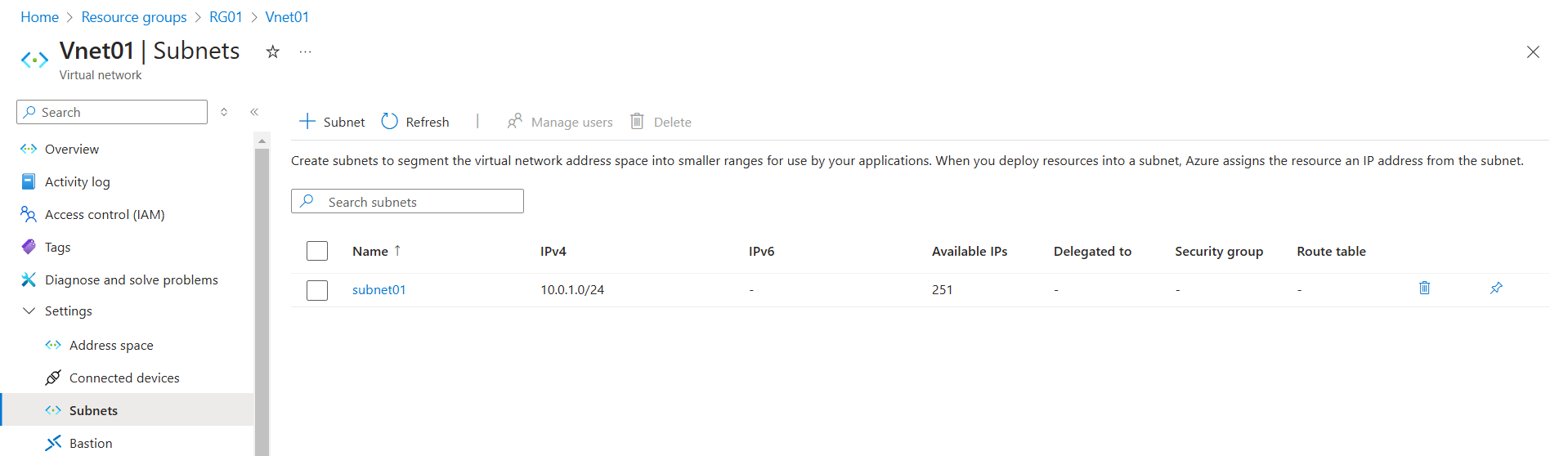
}

**Fig:** resource block.

Then do the terraform apply command. I t will execute or creates the resource hierarchal.







**Fig**: resource group,Vnet and subnet are created hierarchal.